Effective Analysis of Attack Trees: A Model-Driven Approach

Rajesh Kumar[1(✉)], Stefano Schivo[1], Enno Ruijters[1], Buğra Mehmet Yildiz[1], David Huistra[1], Jacco Brandt[1], Arend Rensink[1], and Mariëlle Stoelinga[1,2]

[1] Formal Methods and Tools, University of Twente, Enschede, The Netherlands
{r.kumar,s.schivo,e.j.j.ruijters,b.m.yildiz,d.j.huistra,
a.rensink,m.i.a.stoelinga}@utwente.nl, j.h.brandt@student.utwente.nl
[2] Department of Software Science, Radboud University, Nijmegen, The Netherlands

Abstract. Attack trees (ATs) are a popular formalism for security analysis, and numerous variations and tools have been developed around them. These were mostly developed independently, and offer little interoperability or ability to combine various AT features.

We present ATTop, a software bridging tool that enables automated analysis of ATs using a model-driven engineering approach. ATTop fulfills two purposes: 1. It facilitates interoperation between several AT analysis methodologies and resulting tools (e.g., ATE, ATCalc, ADTool 2.0), 2. it can perform a comprehensive analysis of attack trees by translating them into timed automata and analyzing them using the popular model checker Uppaal, and translating the analysis results back to the original ATs. Technically, our approach uses various metamodels to provide a unified description of AT variants. Based on these metamodels, we perform model transformations that allow to apply various analysis methods to an AT and trace the results back to the AT domain. We illustrate our approach on the basis of a case study from the AT literature.

1 Introduction

Formal methods are often employed to support software engineers in particularly complex tasks: model-based testing, type checking and extended static checking are typical examples that help in developing better software faster. This paper is about the reverse direction: showing how software engineering can assist formal methods in developing complex analysis tools.

More specifically, we reap the benefits of model-driven engineering (MDE) to design and build a tool for analyzing attack trees (ATs). ATs [25,31] are a popular formalism for security analysis, allowing convenient modeling and analysis of complex attack scenarios. ATs have become part of various system engineering frameworks, such as UMLsec [16] and SysMLsec [27].

Attack trees come in a large number of variations, employing different security attributes (e.g., attack time, costs, resources, etc.) as well as modeling constructs (e.g., sequential vs. parallel execution of scenarios). Each of these variations comes with its own tooling; examples include ADTool [12], ATCalc [2],

A. Russo and A. Schürr (Eds.): FASE 2018, LNCS 10802, pp. 56–73, 2018.
https://doi.org/10.1007/978-3-319-89363-1_4

and Attack Tree Evaluator [5]. This "jungle of attack trees" seriously hampers the applicability of ATs, since it is impossible or very difficult to combine different features and tooling. This paper addresses these challenges and presents ATTop[1], a software tool that overarches existing tooling in the AT domain.

In particular, the main features of ATTop are (see Fig. 1):

1. *A unified input format that encompasses the known AT features.* We have collected these features in one comprehensive metamodel. Following MDE best practices, this metamodel is extensible to easily accommodate future needs.
2. *Systematic model transformations.* Many AT analysis methods are based on converting the AT into a mathematical model that can be analyzed with existing formal techniques, such as timed automata [11,23], Bayesian networks [13], Petri nets [8], etc. An important contribution of our work is to make these translations more systematic, and therefore more extensible, maintainable, reusable, and less error-prone.
   To do so, we again refer to the concepts of MDE and deploy *model transformations.* We deploy two categories here: so-called *horizontal* transformations achieve interoperability between existing tools. *Vertical* transformations interpret a model via a set of semantic rules to produce a mathematical model to be analyzed with formal methods.
3. *Bringing the results back to the original domain.* When a mathematical model is analyzed, the analysis result is computed in terms of the mathematical model, and not in terms of the original AT. For example, if AT analysis is done via model checking, a trace in the underlying model (i.e., transition system) can be produced to show that, say, the cheapest attack costs $100. What security practitioners need, however, is a path or attack vector in the original AT. This interpretation in terms of the original model is achieved by a vertical model transformation in the inverse direction, from the results as obtained in the analysis model back into the AT domain.

These features make ATTop a *software bridging tool*, acting as a bridge between existing AT languages, and between ATs and formal languages.

Our Contributions. The contributions of this paper include:

- a full-fledged tool based on MDE, which allows for high maintainability and extensibility;
- a unified input format, enabling interoperability between different AT dialects;
- systematic use of model transformations; which increases reusability while reducing error likelihood;
- a complete cycle from AT to formal model and back, allowing domain experts to profit from formal methods without requiring specific knowledge.

Overview of Our Approach. Figure 1 depicts the general workflow of our approach. It shows how ATTop acts as a bridge between different languages and

[1] Available at https://github.com/utwente-fmt/attop.

formalisms. In particular, thanks to horizontal transformations, ATTop makes it possible to use ATs described in different formats, both as an input to other tools and as an input to ATTop itself. In the latter case, vertical transformations are used in order to deal with UPPAAL as a back-end tool without exposing ATTop's users to the formal language of timed automata.

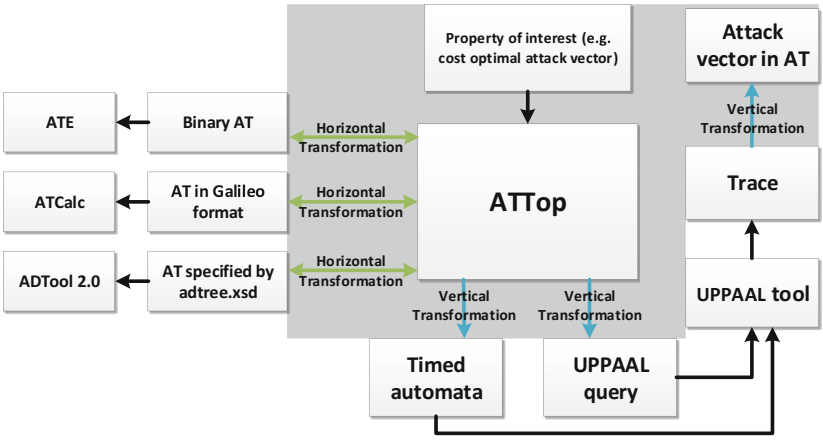

Fig. 1. Overview of our approach, showing the contributions of the paper in the gray rectangle. Here ATE, ATCalc, ADTool 2.0 are different attack tree analysis tools, each with its own input format. ATTop allows these tools to be interoperable (horizontal model transformations, see Sect. 4.1). ATTop also provides a much more comprehensive AT analysis by automatic translation of attack trees into timed automata and using UPPAAL as the back-end analysis tool (vertical transformations, see Sect. 4.2).

Related Work. A large number of AT analysis frameworks have been developed, based on lattice theory [18], timed automata [11,21,23], I/O-IMCs [3,22], Bayesian networks [13], Petri nets [8], stochastic games [4,15], etc. We refer to [20] for an overview of AT formalisms. Surprisingly, little effort has been made to provide a security practitioner with a generic tool that integrates the benefits of all these analysis tools.

The use of model transformations with UPPAAL was explored in [29] for a range of different formalisms; the UPPAAL metamodel that was presented there is the one we use in ATTop. A related approach for fault trees was proposed in [28]. In [14], the authors manually translate UML sequence diagrams into timed automata models to analyze timeliness properties of embedded systems. In [1], the OpenMADS tool is proposed that takes the input of SysML diagrams and UML/MARTE annotations and automatically translates these into deterministic and stochastic Petri nets (DSPNs); however, no model-driven engineering technique was applied.

Organization of the Paper. In Sect. 2, we describe the background. Section 3 presents the metamodels we use in ATTop, while the model transformations are

described in Sect. 4. Section 5 describes the features of ATTop, and in Sect. 6 we show the results of our case study using ATTop. Finally, we conclude the paper in Sect. 7.

2 Background

2.1 Attack Trees in the Security Domain

Modern enterprises are ever growing complex socio-technical systems comprised of multiple actors, physical infrastructures, and IT systems. Adversaries can take advantage of this complexity, by exploiting multiple security vulnerabilities simultaneously. Risk managers, therefore, need to predict possible attack vectors, in order to combat them. For this purpose, attack trees are a widely-used formalism to identify, model, and quantify complex attack scenarios.

Attack trees (ATs) were popularized by Schneier through his seminal paper in [31] and were later formalized by Mauw in [25]. ATs show how different attack steps combine into a multi-stage attack scenario leading to a security breach. Due to the intuitive representation of attack scenarios, this formalism has been used in both academia and industry to model practical case studies such as ATMs [10], SCADA communication systems [7], etc. Furthermore, the attack tree formalism has also been advocated in the Security Quality Requirements Engineering (SQUARE) [26] methodology for security requirements.

Example 1. Figure 2 shows an example AT (adapted from [36]) modeling the compromise of an Internet of Things (IoT) device.

At the top of the tree is the event `compromise_IoT_device`, which is refined using *gates* until we reach the atomic steps where no further refinement is desired (the leaves of the tree). The top gate in Fig. 2 is a SAND (*sequential AND*)-gate denoting that, in order for the attack to be successful, the children of this gate must be executed sequentially from left to right. In the example, the attacker first needs to successfully perform `access_home_network`, then `exploit_software_vulnerability_in_IoT_device`, and then `run_malicious_script`. The AND-gate at `access_home_network` represents that both `gain_access_to_private_networks` and `get_credentials` must be performed, but these can be performed in any order, possibly in parallel. Similarly, the OR gate at `gain_access_to_private_networks` denotes that its children `access_LAN` and `access_WLAN` can be attempted in parallel, but only one needs to succeed for a successful attack.

Traditionally, each leaf of an attack tree is decorated with a single attribute, e.g., the probability of successfully executing the step, or the cost incurred when taking this step. The attributes are then combined in the analysis to obtain metrics, such as the probability or required cost of a successful attack [19].

Over the years, the AT formalism has been enriched both structurally (e.g., adding more logical gates, countermeasures, ordering relationships; see [20] for

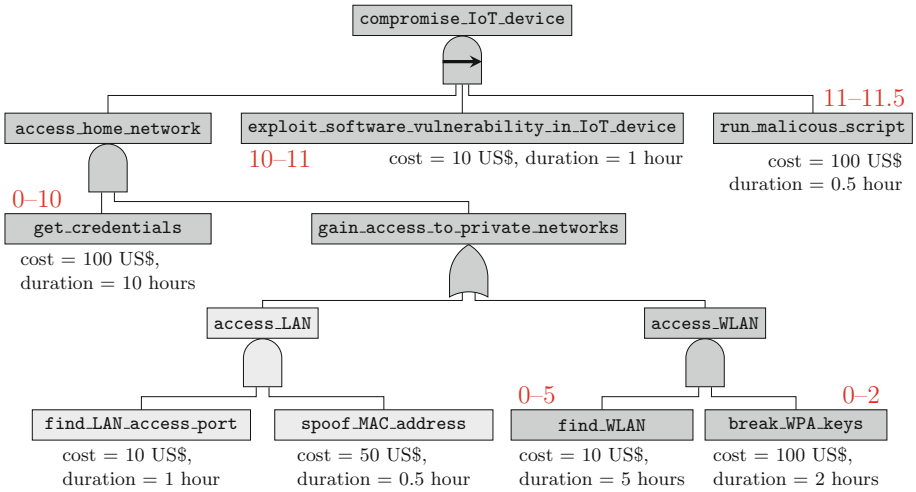

Fig. 2. Attack tree modeling the compromise of an IoT device. Leaves are equipped with the cost and time required to execute the corresponding step. The parts of the tree attacked in the cheapest successful attack are indicated by a darker color, with start and end times for the steps in this cheapest attack denoted in red (times correspond to the scenario in Fig. 11). (Color figure online)

an overview) and analytically (e.g., multi-attribute analysis, time- and cost-optimal analysis). This has resulted in a large number of tools (ADTool 2.0 [12], ATCalc [5], ATE [2], etc.), each with their own analysis technique.

Such a wide range of tools can be useful for a security practitioner to perform different kinds of analyses of attack trees. However, this requires preparing the AT for each tool, as each one has its own input format. To overcome the difficulty of orchestrating all these different tools, we propose one tool—ATTop—to allow specification of ATs combining features of multiple formalisms and to support analysis of such ATs by different tools without duplicating it for each tool.

2.2 Model-Driven Engineering

Model-driven engineering (MDE) is a software engineering methodology that treats models not only as documentation, but also as first-class citizens, to be directly used in the engineering processes [32]. In MDE, a *metamodel* (also referred to as a *domain-specific language*, DSL) is specified as a model at a more abstract level to serve as a language for models [33]. A metamodel captures the concepts of a particular domain with the permitted structure and behavior, to which models must adhere. Typically, metamodels are specified in class diagram-like structures.

MDE provides interoperability between domains (and tools and technologies in these domains) via *model transformations*. The concept of model transformation is shown in Fig. 3. Model transformations map the elements of a source

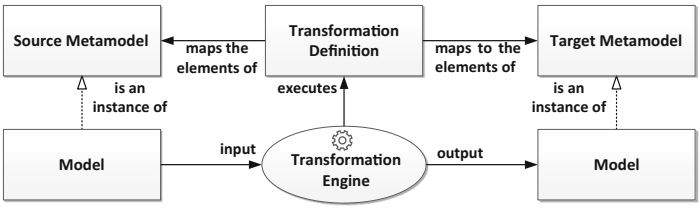

Fig. 3. The concept of *model transformation*

metamodel to the elements of a target metamodel. This mapping is described as a transformation definition, using a language specifically designed for this purpose. The transformation engine executes the transformation definition on the input model and generates an output model.

Adaptation of MDE provides various benefits [30,34,37], specifically:

1. *Empowering domain experts with abstraction:* With the introduction of metamodels and related tooling, domain experts can focus on modeling in the domain; while the technical problems below the modeling level, such as low-level implementation details are abstracted away from the domain experts.
2. *Higher level of reusability:* The models, metamodels and the tools based on them are high-level artifacts that can be reused by many projects targeting similar domains. Such reuse increases productivity and quality of the final product since the reused units are maintained and improved continuously.
3. *Interoperability:* There can be various tools and technologies used in a domain, each having its own I/O formats. Model transformations provide interoperability between these tools and technologies.

There are a number of tools available for realizing MDE. In this paper, we have used the Eclipse Modeling Framework (EMF) [35], which is a state-of-the-art tool developed to this aim. EMF provides the *Ecore* format for defining the metamodels and has many plug-ins to support the various functionalities related to MDE. The model transformations we present in this paper were implemented using the Epsilon Transformation Language (ETL) [17], which is one of the domain-specific languages provided by the Epsilon framework. We have chosen ETL since it is an easy-to-use language and allows users to inherit, import and reuse other Epsilon modules, which increases reusability. We use Java to select and execute the ETL transformations.

3 Metamodels for Attack Tree Analysis

ATTop uses three different metamodels to represent the attack tree domain concepts, all defined in the Ecore format. These are shown in Figs. 4, 5 and 6, in a notation similar to that of UML class diagrams. They show the domain classes and edges representing associations between classes. Edges denote references (→), containment (—◆), or supertype (—▷) relations. Multiplicities are denoted between square brackets (e.g., [0..*] for unrestricted multiplicity).

1. The **AT metamodel (ATMM)**, unifies several extensions of the attack tree formalism including traditional attack trees [25,31], attack-defense trees [18], defense trees [6], etc. It consists of two parts: the *Structure metamodel* and the *Values metamodel*. Below we describe the most important design choices that led to the ATMM:
   - The ATMM represents the core, generic concepts of ATs, resulting in a minimal (and thus clean) metamodel that a domain expert can easily read, understand and use to create models.
   - The ATMM provides a lot of flexibility in specifying the relevant concepts by using string names and generic values. Concepts such as the Connector and the Edge are specified as abstract entities with a set of concrete instances. Therefore, new connectors and edges can easily be added to the metamodel without breaking existing model instances. The metamodel is designed to have good support for model operations, such as traversal of the AT models. From a node, any other node can be reached directly or indirectly following references.
   - The ATMM node and tree attributes offer convenient and generic methods for supporting the results of analysis tools. This allows us to translate results from a formal tool back into the AT domain and associate them to the original AT model (see Sect. 4.4).
2. The **query metamodel** formalizes the security queries to be analyzed over attack trees. We support both qualitative queries (i.e., properties such as feasibility of attack) and quantitative queries (i.e., security metrics such as probability of successful attack, cheapest attack, etc.).
3. The **scenario metamodel** represents attack scenarios (a.k.a. attack vectors) consisting of the steps leading to, e.g., the cheapest, fastest, or most damaging security breaches.

Below we discuss these metamodels in more detail.

1. AT Metamodel (ATMM). The ATMM metamodel is a combination of two separate metamodels, one representing the attack tree structure (*Structure metamodel*, Fig. 4 left) and the other representing the attack tree attributes (*Values metamodel*, Fig. 4 right). This separation allows us to consider different attack scenarios modeled via the same attack tree, but decorated with different attributes. For example, it is easy to define attribute values based on the attacker type: script kiddie, malicious insider, etc. may be all be interested in the same asset, but each of them possesses different access privileges and is equipped with different resources.

Structure Metamodel. The structure model, depicted in Fig. 4 on the left, represents the structure of the attack tree. Its main class AttackTree contains a set of one or more Nodes, as indicated by the containment arrow between AttackTree and Node. One of these nodes is designated as the root of the tree, denoted by the root reference. Each Node is equipped with an id, used as a reference during transformation processes. Furthermore, each node has a (possibly empty) list of its parents and children, which allows to easily traverse the AT. A node may have a connector, i.e., a *gate* such as AND, OR, SAND (sequential-AND), etc.

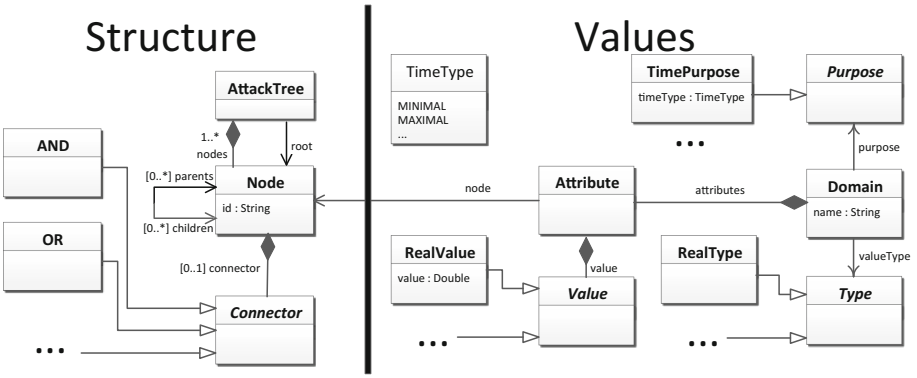

Fig. 4. The ATMM metamodel separated into the structure and values metamodels. Some connectors, types, and purposes are omitted for clarity and denoted by ellipses.

In addition to the structure specified by the metamodel, some constraints can be used to ensure that a model is a valid attack tree. For example, the tree cannot contain cycles, the nodes must form a connected graph, etc. These constraints are separately formulated in the Epsilon Validation Language (EVL [17]). An example of such a constraint is shown in Listing 1.

Values Metamodel. The Values metamodel (Fig. 4, right side) describes how values are attributed to nodes (arrow from Attribute on the right to Node on the left). Each Attribute contains exactly one Value, which can be of various (basic or complex) types: For example, RealValue is a type of Value that contains real (Double) numbers. A Domain groups all those attributes that have the same Purpose. By separating the purpose of attributes from their data type, we can use basic data types (integer, boolean, real number) for different purposes: For example, a real number (RealType) can be used in a Domain named "Maximum Duration", where the purpose is a TimePurpose with timeType = MAXIMAL. A RealType number could also be used in a different Domain, say "Likelihood of attack" with the purpose to represent a probability (ProbabilityPurpose, not shown in the diagram). Thanks to the flexibility of this construct, the set of available domains is easily extensible.

```
1  context ATMM!AttackTree {
2      constraint OneAndOnlyOneChildWithoutParents {
3          check : ATMM!Node.allInstances.select(n|n.parents.size() == 0).size() = 1
4          and self.root = ATMM!Node.allInstances.select(n|n.parents.size() == 0).first()
5      }
6  }
```

Listing 1. Constraint specifying that the root node is the only node in an ATMM AT with no parents.

2. Query Metamodel. Existing attack tree analysis tools such as ATE, ATCalc, ADTool 2.0, etc. support only a limited set of queries, lacking the flexibility to customize one's own security queries. Using the MDE approach, we have developed the Query metamodel shown in Fig. 5. This allows a security practitioner to ask a wide range of qualitative and quantitative metrics over a wide range of attributes such as cost, time, damage, etc.

Using this metamodel in ATTop, a security practitioner can ask all the security queries available in the aforementioned tools. Furthermore, the metamodel offers a more comprehensive set of security queries where users can tailor their own security queries. For example, it is possible to ask whether a successful attack can be carried out within 10 days and without spending more than $900.

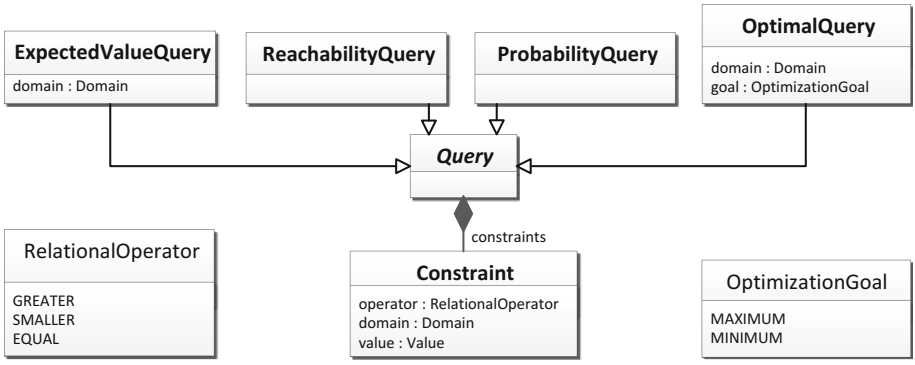

Fig. 5. The query metamodel. The types 'Domain' and 'Value' refer to the classes of the ATMM metamodel (Fig. 4).

The main component of the query metamodel is the element named Query. A query can be one of the following:

- Reachability, i.e., Is it feasible to reach the top node of an attack tree? Supported by every tool.
- Probability, i.e., What is the probability that a successful attack occurs? Supported by every tool.
- ExpectedValue, i.e., What is the expected (average) value of a given quantity over all possible attacks? Supported by ATTop.
- Optimality, i.e., Which is the attack that is optimal w.r.t. a given attribute (e.g., time or cost)? Supported by ATE, ADTool 2.0, ATTop.

Furthermore, a query can be framed by combining one of the above query types with a set of Constraints over the AT attributes. A Constraint is made of a RelationalOperator, a Value and its Domain. For example, the constraint "within 10 days" is expressed with the SMALLER RelationalOperator, a Value of 10, and the Domain of "Maximum Duration".

3. Scenario Metamodel. ATTop is geared to provide different results: some of which are numeric, like the probability to execute attack, the maximum cost to execute an attack, etc. Other results contain qualitative information such as an attack vector, which is a partially ordered set of basic attack steps resulting in the compromise of an asset under a given set of constraints (for example, incurring minimum cost). In order to properly trace back the qualitative output to the original attack tree, we use the Scenario metamodel (see Fig. 6).

The Scenario metamodel is used to represent attack vectors. In our context, we consider an attack vector to be a Schedule where there is only one Executor, which we name "Attacker". The sequence of Tasks appearing in a Scenario are then interpreted as the sequence of the attack steps the Attacker needs to carry out in order to reach their objective. Each attack step is actually a node of the original AT, and is represented as an Executable whose name corresponds to the id of the original Node. Timing information contained in each Task describes the start (startTime) and end (endTime) time points for each attack step. Note that an attack can start but not end before the objective is reached (multiplicity "1" for startTime and "0..1" for endTime).

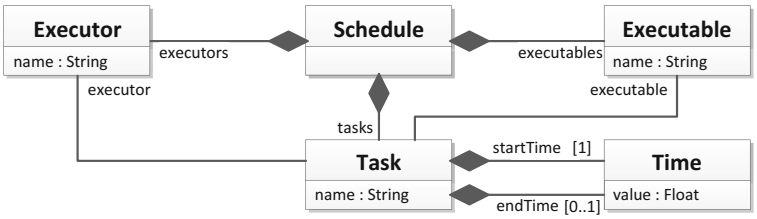

Fig. 6. The Scenario metamodel from [29]. In the context of ATs, all instances of this metamodel will have only one Executor, the Attacker; Executables represent attack steps (i.e. Nodes from the AT), while a Scenario is known as an attack vector.

4 Model Transformations

ATTop supports *horizontal* and *vertical* model transformations. Figure 7 illustrates the difference between these. *Horizontal transformations* convert one model into another that conforms to the same metamodel, e.g., a transformation from one AT analysis tool to another (where the models of both tools are represented in the ATMM metamodel). *Vertical transformations* transform a model into another that conforms to a different metamodel, e.g., the transformation from an AT into a timed automaton. A key feature of ATTop is that it also provides vertical transformations in the reverse direction: analysis results (e.g., traces produced by Uppaal) are interpreted in terms of the original attack tree model.

4.1 Horizontal Transformations: Unifying Dialects of Attack Trees

One of the goals of applying the model-driven approach is to facilitate interoperation between different tools. To this end, we provide transformations to and from the file formats of ADTool 2.0 [12], Attack Tree Evaluator (ATE) [5], and ATCalc [2].

Due to the different features supported by the various tools, not all input formalisms can be converted to any other format preserving all semantics. For example, ATCalc performs only timing analysis, while ADTool can also perform cost analysis of untimed attack trees. In such cases, the transformations convert whatever information is supported by their output format, omitting unsupported features. As the ATMM metamodel unifies the features of all the listed tools, transformations into this metamodel are lossless.

Example 2. ATE Transformation. The Attack Tree Evaluator [5] tool can only process binary trees. Using a simple transformation, we can transform any instance of the ATMM into a binary tree. A simplified version of this transformation, written in ETL, is given in Listing 2. This transformation is based on a recursive method that traverses the tree. For every node with more than two children, it nests all but the first child under a new node until no more than two children remain.

4.2 Vertical Transformations: Analyzing ATs via Timed Automata

Thus far we have described the transformations to and from dedicated tools for attack trees. In this section we introduce a vertical transformation which we use in ATTop to translate attack trees into the more general-purpose formalism of timed automata (TA). Specifically, we provide model transformations to TAs that can be analyzed by the Uppaal tool to obtain the wide range of qualitative and quantitative properties supported by the query metamodel.

Our transformation targets the Uppaal metamodel described in [29]. It transforms each element of the attack tree (i.e., each gate and basic attack step)

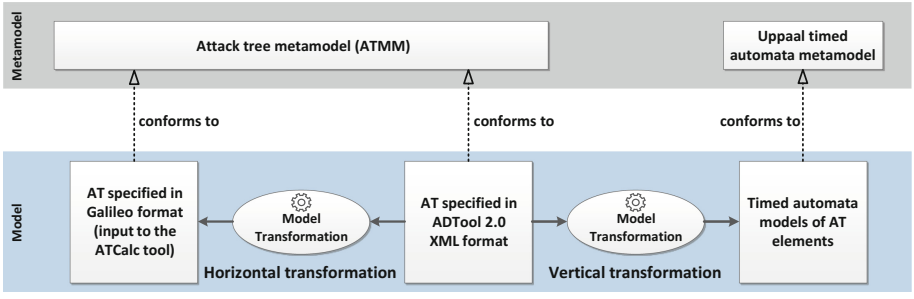

Fig. 7. Examples of *horizontal* and *vertical* model transformations.

```
var structure := AttackTree.all.first();
structure.Root.NodeToBinary();

operation Node NodeToBinary(){
    if(self.Children.size()>2){
        var newNode = new Node();
        newNode.Parents.add(self);
        structure.Nodes.add(newNode);

        var replaceNodes := self.Children.excluding(self.Children.first());
        newNode.Children := replaceNodes;
        self.Children.removeAll(replaceNodes);
        self.Children.add(newNode);
    }
    for(child in self.Children)
        child.NodeToBinary();
}
```

Listing 2. Transformation of an ATMM attack tree to a binary AT

into a timed automaton. These automata communicate via signals and together describe the behavior of the entire tree. For example, Fig. 8 shows the timed automaton obtained by transforming an attack step with a deterministic time to execute of 5 units.

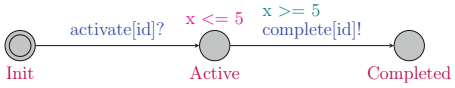

Fig. 8. Example of a timed automaton modeling a basic attack step with a fixed time to execute of 5 units.

Depending on the features of the model and the desired property to be analyzed, the output of the transformation can be analyzed by different extensions of Uppaal. For example, Uppaal CORA supports the analysis of cost-optimal queries, such as "What is the lowest cost an attacker needs to incur in order to complete an attack", while Uppaal-SMC supports statistical model checking, allowing the analysis of models with stochastic times and probabilistic attack steps with queries such as "What is the probability that an attacker successfully completes an attack within one hour". The advantages of Uppaal CORA's exact results come at the cost of state space explosion, which limits the applicability of this approach for larger problems. On the other hand, the speed and scalability of the simulation-based Uppaal-SMC are countered by approximated results and the unavailability of (counter-)example traces.

4.3 Query Transformation: From Domain-Specific to Tool-Specific

ATTop aims to enable the analysis of ATs also by users that are less familiar with the underlying tools. One challenge for such a user is that every tool has its own method to specify what property of the AT should be computed.

Section 3 describes our metamodel for expressing a wide range of possible queries, and we now transform such queries to a tool-specific format. Many tools

support only a single query (e.g., ATE [5] only supports Pareto curves of cost vs. probability), in which case no transformation is performed but ATTop only allows that single query as input.

The Uppaal tool is an example of a tool supporting many different queries. After transforming the AT to a timed automaton (cf. Sect. 4.2), we transform the query into the textual formula supported by Uppaal. The basic form of this formula is determined by the query type (e.g., a ReachabilityQuery will be translated as "`E<> toplevel.completed`", which asks for the existence of a trace that reaches the top level event), while constraints add additional terms limiting the permitted behavior of the model. By using an Uppaal-specific metamodel for its query language linked to the TA metamodel, our transformation can easily refer to the TA elements that correspond to converted AT elements.

4.4 Result Transformation: From Tool-Specific to Domain-Specific

Analyses done with a back-end tool produce results that may only be immediately understandable to an expert in that tool. An important feature of ATTop to ease its use by non-experts, is that it provides interpretations of these results in terms of the original AT.

For example, given an attack tree whose leaves are annotated with (time-dependent) costs, Uppaal can produce a trace showing the cheapest way to reach a security breach (optionally within a specified time bound). This trace is given in a textual format, with many details that are irrelevant to a security analyst. It is much easier to understand this scenario when shown in terms of the attack tree (for example, Fig. 11 is a scenario described by several pages of Uppaal output). This is exactly the purpose of having reverse transformations: Uppaal's textual traces are automatically parsed by ATTop, generating instances of the Trace metamodel described in [29]. To do so, the transformation from ATMM to Uppaal retains enough information to trace identifiers in the Uppaal model back to the elements of the AT. When parsing the trace, ATTop extracts only the relevant events (e.g., the starts and ends of attack steps) and related information (e.g., time). This information is then stored as an instance of the Scenario metamodel described in Sect. 3.

In the generated Schedule, attack steps are represented as Executables, while Tasks indicate the start and finish time of each attack step, thus describing the attack vector. Only one Executor is present in any attack vector produced by this transformation, and that is the Attacker. An example of such a generated schedule can be seen in Fig. 11.

5 Tool Support

We have developed the tool ATTop to enable users to easily use the transformations described in this paper, without requiring knowledge of the underlying techniques or formalisms. ATTop automatically selects which transformations to apply based on the available inputs and desired outputs. For example, if the user provides an ADTool input and requests an Uppaal output,

ATTop will automatically first execute the transformation from ADTool to the ATMM, and then the transformation from ATMM to UPPAAL.

Users operate the tool by specifying input files and their corresponding languages, and the desired output files and languages. ATTop then performs a search for the shortest sequence of transformations achieving the desired outputs from the inputs. For example, Fig. 9 shown the tool's main screen, where the user has provided an input AT in Galileo format. The user can now choose between different queries and analysis engines.

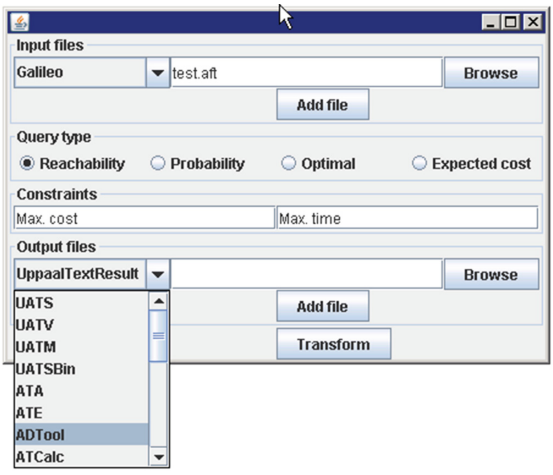

Fig. 9. Screenshot of ATTop's main screen, allowing input file selection, query specification, and output selection.

6 Case Study

As a case study we use the example annotated attack tree given in Fig. 2. We apply ATTop to automatically compute several qualitative and quantitative security metrics. Specifically, we apply a horizontal transformation to convert the model from the ATCalc format to that accepted by ADTool 2.0, and a vertical transformation to analyze the model using UPPAAL.

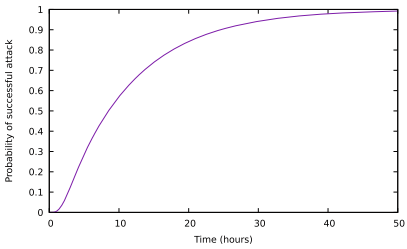

Fig. 10. ATCalc plot showing probability of successful attack over time

We specify the AT in the Galileo format as accepted by ATCalc. Analysis with ATCalc yields a graph of the probability of a successful attack over time, as shown in Fig. 10. Next, we would like to determine the minimal cost of a successful attack, which ATCalc cannot provide. Therefore, we use ATTop to transform the AT to the ADTool 2.0 format, and use ADTool 2.0 to compute the minimal cost (yielding \$270).

Next, we perform a more comprehensive timing analysis using the vertical transformation described in Sect. 4.2. We use ATTop to transform the AT to a timed automaton that can be analyzed using the UPPAAL tool. We also transform a query (OptimalityQuery asking for minimal time) to the corresponding UPPAAL query. Combining these, we obtain a trace for the fastest successful attack, which ATTop transforms into a scenario in terms of the AT as described in Sect. 4.3.

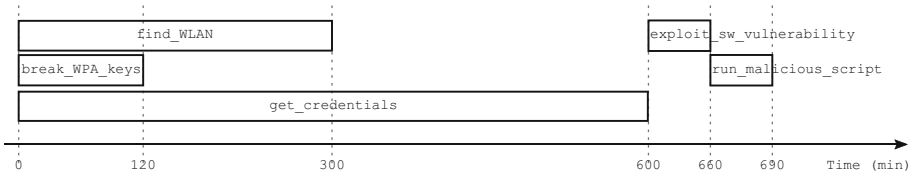

Fig. 11. Scenario of fastest attack as computed by UPPAAL . The executed steps and their start–end times are also shown in Fig. 2.

The resulting scenario is shown in Fig. 11. Running the whole process, including the transformations and the analysis with UPPAAL, took 6.5 s on an Intel® Core™ i7 CPU 860 at 2.80 GHz running Ubuntu 16.04 LTS.

7 Conclusions

We have presented a model-driven approach to the analysis of attack trees and a software bridging tool—ATTop—implementing this approach. We support interoperability between different existing analysis tools, as well as our own analysis using the popular tool UPPAAL as a back-end engine.

Formal methods have the advantage of being precise, unambiguous and systematic. A lot of effort is spent on their correctness proofs. However, these benefits are only reaped if the tools supporting formal analysis are also correct. To the best of our knowledge, this work is among the first to apply the systematic approach of MDE to the development of formal analysis tools.

Through model-driven engineering, we have developed the attack tree metamodel (ATMM) with support for the many extended formalisms of attack trees, integrating most of the features of such extensions. This unified metamodel provides a common representation of attack trees, allowing easy transformations from and to the specific representations of individual tools such as ATCalc [2] and ADTool [12]. The metamodels for queries and schedules facilitate a user-friendly interface, obtaining relevant questions and presenting results without needing expert knowledge of the underlying analysis tool.

We have presented our approach specifically for attack trees, but we believe it can be equally fruitful for different formalisms and tools as well (e.g. PRISM [24], STORM [9]) by using different metamodels and model transformations. We thus expect our approach to be useful in the development of other tools that bridge specialized domains and formal methods.

Acknowledgments. This research was partially funded by STW and ProRail under the project ArRangeer (grant 12238), STW, TNO-ESI, Océ and PANalytical under the project SUMBAT (13859), STW project SEQUOIA (15474), NWO projects BEAT (612001303) and SamSam (628.005.015), and EU project SUCCESS (102112).

References

1. Andrade, E.C., Alves, M., Matos, R., Silva, B., Maciel, P.: OpenMADS: an open source tool for modeling and analysis of distributed systems. In: Bitsch, F., Guiochet, J., Kaâniche, M. (eds.) SAFECOMP 2013. LNCS, vol. 8153, pp. 277–284. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40793-2_25
2. Arnold, F., Belinfante, A., Van der Berg, F., Guck, D., Stoelinga, M.: DFTCalc: a tool for efficient fault tree analysis. In: Bitsch, F., Guiochet, J., Kaâniche, M. (eds.) SAFECOMP 2013. LNCS, vol. 8153, pp. 293–301. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40793-2_27
3. Arnold, F., Guck, D., Kumar, R., Stoelinga, M.: Sequential and parallel attack tree modelling. In: Koornneef, F., van Gulijk, C. (eds.) SAFECOMP 2015. LNCS, vol. 9338, pp. 291–299. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24249-1_25
4. Aslanyan, Z., Nielson, F., Parker, D.: Quantitative verification and synthesis of attack-defence scenarios. In: Computer Security Foundations (CSF), pp. 105–119 (2016). https://doi.org/10.1109/CSF.2016.15
5. Aslanyan, Z.: Attack Tree Evaluator, developed for EU project TREsPASS, Technical University of Denmark. https://vimeo.com/145070436
6. Bistarelli, S., Fioravanti, F., Peretti, P., Santini, F.: Evaluation of complex security scenarios using defense trees and economic indexes. J. Exp. Theor. Artif. Intell. **24**(2), 161–192 (2012). https://doi.org/10.1080/13623079.2011.587206
7. Byres, E.J., Franz, M., Miller, D.: The use of attack trees in assessing vulnerabilities in SCADA systems. In: Proceedings of Infrastructure Survivability Workshop. IEEE (2004)
8. Dalton, G.C.I., Mills, R.F., Colombi, J.M., Raines, R.A.: Analyzing attack trees using generalized stochastic petri nets. In: 2006 IEEE Information Assurance Workshop, pp. 116–123, June 2006. https://doi.org/10.1109/IAW.2006.1652085
9. Dehnert, C., Junges, S., Katoen, J.-P., Volk, M.: A Storm is coming: a modern probabilistic model checker. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10427, pp. 592–600. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63390-9_31
10. Fraile, M., Ford, M., Gadyatskaya, O., Kumar, R., Stoelinga, M., Trujillo-Rasua, R.: Using attack-defense trees to analyze threats and countermeasures in an ATM: a case study. In: Horkoff, J., Jeusfeld, M.A., Persson, A. (eds.) PoEM 2016. LNBIP, vol. 267, pp. 326–334. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48393-1_24
11. Gadyatskaya, O., Hansen, R.R., Larsen, K.G., Legay, A., Olesen, M.C., Poulsen, D.B.: Modelling attack-defense trees using timed automata. In: Fränzle, M., Markey, N. (eds.) FORMATS 2016. LNCS, vol. 9884, pp. 35–50. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-44878-7_3
12. Gadyatskaya, O., Jhawar, R., Kordy, P., Lounis, K., Mauw, S., Trujillo-Rasua, R.: Attack trees for practical security assessment: ranking of attack scenarios with ADTool 2.0. In: Agha, G., Van Houdt, B. (eds.) QEST 2016. LNCS, vol. 9826, pp. 159–162. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-43425-4_10
13. Gribaudo, M., Iacono, M., Marrone, S.: Exploiting Bayesian networks for the analysis of combined attack trees. In: Proceedings of PASM. ENTCS, vol. 310, pp. 91–111 (2015). https://doi.org/10.1016/j.entcs.2014.12.014

14. Hendriks, M., Verhoef, M.: Timed automata based analysis of embedded system architectures. In: Proceedings of 20th International Conference on Parallel and Distributed Processing (IPDPS), p. 179. IEEE (2006). https://doi.org/10.1109/IPDPS.2006.1639422
15. Hermanns, H., Krämer, J., Krčál, J., Stoelinga, M.: The value of attack-defence diagrams. In: Piessens, F., Viganò, L. (eds.) POST 2016. LNCS, vol. 9635, pp. 163–185. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49635-0_9
16. Jürjens, J.: UMLsec: extending UML for secure systems development. In: Jézéquel, J.-M., Hussmann, H., Cook, S. (eds.) UML 2002. LNCS, vol. 2460, pp. 412–425. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45800-X_32
17. Kolovos, D., Rose, L., García-Domńguez, A., Paige, R.: The Epsilon Book (2016). http://www.eclipse.org/epsilon/doc/book
18. Kordy, B., Mauw, S., Radomirović, S., Schweitzer, P.: Foundations of attack–defense trees. In: Degano, P., Etalle, S., Guttman, J. (eds.) FAST 2010. LNCS, vol. 6561, pp. 80–95. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19751-2_6
19. Kordy, B., Mauw, S., Schweitzer, P.: Quantitative questions on attack–defense trees. In: Kwon, T., Lee, M.-K., Kwon, D. (eds.) ICISC 2012. LNCS, vol. 7839, pp. 49–64. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37682-5_5
20. Kordy, B., Pièvre-Cambacédès, L., Schweitzer, P.: DAG-based attack and defense modeling: don't miss the forest for the attack trees. Comput. Sci. Rev. **13–14**, 1–38 (2014). https://doi.org/10.1016/j.cosrev.2014.07.001
21. Kumar, R., Stoelinga, M.: Quantitative security and safety analysis with attack-fault trees. In: Proceedings of IEEE 18th International Symposium on High Assurance Systems Engineering (HASE), pp. 25–32, January 2017. https://doi.org/10.1109/HASE.2017.12
22. Kumar, R., Guck, D., Stoelinga, M.: Time dependent analysis with dynamic counter measure trees. In: Proceedings of 13th Workshop on Quantitative Aspects of Programming Languages (QAPL) (2015). http://arxiv.org/abs/1510.00050
23. Kumar, R., Ruijters, E., Stoelinga, M.: Quantitative attack tree analysis via priced timed automata. In: Sankaranarayanan, S., Vicario, E. (eds.) FORMATS 2015. LNCS, vol. 9268, pp. 156–171. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-22975-1_11
24. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_47
25. Mauw, S., Oostdijk, M.: Foundations of attack trees. In: Won, D.H., Kim, S. (eds.) ICISC 2005. LNCS, vol. 3935, pp. 186–198. Springer, Heidelberg (2006). https://doi.org/10.1007/11734727_17
26. Mead, N.: SQUARE Process (2013). https://buildsecurityin.us-cert.gov/articles/best-practices/requirements-engineering/square-process
27. Roudier, Y., Apvrille, L.: SysML-Sec: a model driven approach for designing safe and secure systems. In: Proceedings of 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD), pp. 655–664 (2015)
28. Ruijters, E., Schivo, S., Stoelinga, M.I.A., Rensink, A.: Uniform analysis of fault trees through model transformations. In: Proceedings of IEEE 63rd Annual Reliability and Maintainability Symposium (RAMS), January 2017. https://doi.org/10.1109/RAM.2017.7889759

29. Schivo, S., Yildiz, B.M., Ruijters, E., Gerking, C., Kumar, R., Dziwok, S., Rensink, A., Stoelinga, M.: How to efficiently build a front-end tool for UPPAAL: a model-driven approach. In: Larsen, K.G., Sokolsky, O., Wang, J. (eds.) SETTA 2017. LNCS, vol. 10606, pp. 319–336. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69483-2_19
30. Schmidt, D.C.: Guest editor's introduction: model-driven engineering. Computer **39**(2), 25–31 (2006). https://doi.org/10.1109/MC.2006.58
31. Schneier, B.: Attack trees. Dr. Dobb's J. **24**(12), 21–29 (1999)
32. da Silva, A.R.: Model-driven engineering: a survey supported by the unified conceptual model. Comput. Lang. Syst. Struct. **43**, 139–155 (2015). https://doi.org/10.1016/j.cl.2015.06.001
33. Sprinkle, J., Rumpe, B., Vangheluwe, H., Karsai, G.: Chapter 3: Metamodelling. In: Giese, H., Karsai, G., Lee, E., Rumpe, B., Schätz, B. (eds.) MBEERTS 2007. LNCS, vol. 6100, pp. 57–76. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16277-0_3
34. Stahl, T., Voelter, M., Czarnecki, K.: Model-Driven Software Development: Technology, Engineering, Management. Wiley, Chichester (2006)
35. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework 2.0, 2nd edn. Addison-Wesley Professional, Reading (2009)
36. Steiner, M., Liggesmeyer, P.: Qualitative and quantitative analysis of CFTs taking security causes into account. In: Koornneef, F., van Gulijk, C. (eds.) SAFECOMP 2015. LNCS, vol. 9338, pp. 109–120. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24249-1_10
37. Völter, M., Stahl, T., Bettin, J., Haase, A., Helsen, S.: Model-Driven Software Development: Technology, Engineering, Management. Wiley, Chichester (2006)